Compas'2022 : Parallélisme / Architecture/ Système
MIS/UPJV - Amiens France, 5-8 juillet 2022

Partage des nœuds E/S entre applications

Alexis Bandet, Francieli Boito, Guillaume Pallez *

Univ. Bordeaux, CNRS, Bordeaux INP, INRIA, LaBRI, UMR 5800, F-33400 Talence, France
alexis.bandet@inria.fr

Résumé
Avec l'arrivée prochaine des machines exascale, l'essor du Big Data et le développement du machine learning dans les centres de calcul l'écart entre les performances de calcul et du système du stockage devient toujours plus une problématique pour maximiser l'efficacité des machines. Ainsi, une sur-couche logicielle et matérielle de transfert des E/S permet de coordonner les flux E/S entre les nœuds de calcul et le système de fichier. En nous appuyant sur une simulation nous proposons une approche de partage des ressources de la couche de transfert d'E/S à l'aide d'algorithmes gloutons, transparente pour l'utilisateur et avec un minimum de connaissance *a priori* des applications. Nos solutions améliorent le stretch de 20% par rapport à un placement aléatoire, qui est le plus proche de ce qui est utilisé sur des vraies machines.

Mots-clés : calcul à hautes performances, E/S, ordonnancement, nœud E/S

1. Introduction

L'émergence de nouvelles applications orientées vers l'apprentissage par ordinateur et le traitement des données mettent à l'épreuve les infrastructures d'entrées et de sorties (E/S) des machines présentes dans les centre de calculs. Sur ces super-calculateur, qui ont toujours plus de capacité de calcul, le système de fichier parallèle (SFP) peuvent se retrouver en difficulté. Afin d'organiser les flux de requêtes d'E/S, une couche logicielle appelée « couche de transfert d'entrées et sortie » est mise en place entre les nœuds de calcul et le système de stockage[8]. Cette couche de transfert entièrement transparent pour l'utilisateur de la machine permet de limiter les interférences et les congestions entre les E/S. Elle permet la mise en place de nombreuses optimisations ont avec pour but d'optimiser l'accès au ressources d'E/S, notamment sous l'angle de l'ordonnancement, de l'agrégation ou du réarragement des requêtes d'E/S[2, 7]. L'affectation des nœuds d'E/S aux applications est souvent effectuée de manière statique, c'est-à-dire que pour N nœuds de traitement et M nœuds d'E/S, chaque nœud d'E/S est connecté à N/M nœuds de traitement, qui ne peuvent pas communiquer avec les autres nœuds d'E/S. Par conséquent, les nœuds d'E/S qu'une application utilisera dépendent de son placement. Néanmoins, d'autres stratégies de placement sont possibles. Dans certains systèmes tels que Tianhe-2 et Sunway TaihuLight, il est possible de configurer le nombre et la nature des nœuds d'E/S auxquels chaque tâche a accès [9, 6].
Se baser sur une corrélation entre les nombres de nœuds d'E/S et de calcul peut être trompeuse car le nombre de nœuds de calcul ne reflète pas le schéma d'accès aux données ni le besoin réel

*. Authors are presented in alphabetical order.

en terme de capacités d'E/S de l'application. L'approche statique peut donc amener à un déséquilibre dans l'utilisation des ressources E/S disponible sur la machine et donc une potentielle perte de performance globale.
Pour cette raison une partie de la recherche s'est concentrée sur la détermination du nombre idéal de nœuds d'E/S à allouer pour l'exécution d'une application avec une connaissance *a priori* des applications[5]. Certains auteurs ont observés que le nombre optimal de nœuds d'E/S d'une application peut dépendre de son schéma d'accès aux données et ont donc proposés des algorithmes pour maximiser la bande passante des E/S[1].
Ces recherches se concentre sur un mapping exclusif des nœuds d'E/S ou sur une configuration de partage décidé en temps réel en cas de surcharge d'un ou plusieurs nœuds afin d'éviter de créer des congestions. Mais si le nombre d'applications est beaucoup plus élevé que le nombre de nœuds E/S disponibles alors un partage de ces nœuds devient inévitable. De plus permettre le partage pour certain nœuds permet la libération de ressources E/S pour les allouer à d'autres applications. Cela permettrait de modérer voir d'annuler les pénalités induites par le partage entre certaines applications. Enfin, certaines applications HPC ont une activité E/S limitée ; leur attribuer de façon exclusive des nœuds E/S serait alors une perte substantielle des capacité d'E/S de la machine.
Notre travail se concentre sur le partage des nœuds d'E/S par les applications, sans connaissance *a priori* de celles-ci et avant leur exécution. Nous allons pour cela mettre en place des algorithmes gloutons avec des heuristiques simples afin d'ordonnancer les applications sur les différents nœuds d'E/S. Le but est de limiter au maximum que deux applications entre en collision, c'est-à-dire qu'elle souhaite faire des E/S en même temps lorsqu'elles ont au moins un nœuds E/S partagé.

2. Partage de nœuds E/S dans des systèmes HPC

Comme discuté dans la section précédente, dans ce travail nous allons nous concentrer sur la partie allocation et partage de nœuds E/S entre applications. Nous présentons ici notre modèle (Section 2.1) et heuristiques (Section 2.2).

2.1. Modèle

Dans notre contexte, une machine peut être décrite par son nombre de nœuds E/S : N. Nous considérons ici que les nœuds ont tous une performance (mesurée en terme de bande passante) homogène que nous normalisons à 1.
Nous considérons que K applications $\{\mathcal{A}_1, \ldots, \mathcal{A}_K\}$ sont placées sur la machine et en concurrence pour l'accès aux N nœuds E/S. En pratique, en général une application $\mathcal{A}_i$ alterne entre des phases d'E/S et des phases de calcul [3, 4]. L'application $\mathcal{A}_i$ utilise n_i nœuds d'E/S lors de ses phases d'E/S. Lors d'une phase de calcul, l'application n'a pas d'impact sur les nœuds E/S. Soit $\bar{T}_c$ le temps total des différentes phases de calcul et $\bar{T}_{io}$ le temps total des différentes phases d'E/S lorsque l'application s'exécute seule sur la plate-forme, nous définissons pour l'application $\mathcal{A}_i$: $r_i = \frac{\bar{T}_{io}}{\bar{T}_{io}+\bar{T}_c}$ le ratio de temps passé à faire de l'E/S en isolation. En pratique et pour le modèle, nous considérons une décomposition du temps unitaire. En isolation, à chaque unité de temps, l'application $\mathcal{A}_i$ a une probabilité r_i de faire des E/S, et une probabilité $1 - r_i$ de faire du calcul.
Le but du travail est de calculer une allocation des nœuds E/S pour chaque application en respectant leur contrainte n_i (c'est à dire qu'on doit allouer n_i nœuds à l'application $\mathcal{A}_i$).
À chaque unité de temps, chaque application peut être dans l'une de ces deux phases : (i) phase de calcul (tous ses processus font du calcul), (ii) phase d'E/S. La phase d'E/S peut être

soit complète (à la fin de l'unité de temps, toutes les E/S seront effectuées), soit incomplète (à la fin de l'unité de temps, certaines E/S resteront à être faites).
À chaque unité de temps, si au moins deux applications sont en phase d'E/S et veulent effectuer une partie de ces E/S sur un même nœud, alors seulement une peut effectuer ses E/S. Les E/S des autres applications utilisant ce nœud sont mises en pauses jusqu'à l'unité de temps suivante, dans ce cas ces applications sont en phase E/S incomplètes. Il existe deux modes pour les phases incomplètes :

— E/S Bloquantes : pour chaque application, si l'un des nœuds d'E/S nécessaire pour effectuer sa phase E/S est utilisé, alors l'application ne fait aucun accès E/S qui sont tous reportés à l'unité de temps suivante.
— E/S Non-bloquantes : si l'un des nœuds d'E/S nécessaire pour effectuer la phase E/S est utilisé, l'application peut effectuer une partie de ses E/S sur ses nœuds disponibles. À l'unité de temps suivante, il restera à l'application à effectuer les E/S qu'elle n'a pas pu faire, sur les nœuds qui n'étaient pas disponibles.

Problème d'optimisation

Pour résoudre ce problème, nous cherchons donc :

1. Trouver une fonction d'allocation
$\sigma : \{1, \ldots, K\} \to \{1, \ldots, N\}^{\mathbb{N}}$ tel que $|\sigma(i)| = n_i$. $\sigma(i) = \{j_1, j_2, \ldots, j_{n_i}\}$ signifie que l'application $\mathcal{A}_i$ utilise les n_i nœuds d'E/S indicés par $\{j_1, j_2, \ldots, j_{n_i}\}$.
2. Trouver une fonction de priorité qui, quand deux applications veulent effectuer des E/S simultanément sur le même nœud d'E/S choisit celle qui effectuera des E/S et celle qui attendra l'unité de temps suivante.

Pour évaluer les choix de fonctions, nous nous concentrons sur les objectifs suivants :

— Le stretch de chaque application : c'est à dire le ratio entre le temps d'exécution étant donné la solution proposée par rapport à son temps d'exécution lorsqu'elle s'exécute en isolation. Cet objectif représente la pénalité subit par l'application liée aux interférences en cas de collision E/S sur notre simulation.
— L'occupation des ressources E/S de la machine : c'est à dire la proportion de temps que chaque nœud E/S travaille.

Pour ces deux grandeurs, il est possible d'évaluer les critères médians, moyens ou maximum.

2.2. Solutions algorithmiques

Pour ce travail préliminaire nous proposons de comparer trois solutions d'allocation entre elles pour obtenir des premiers éléments d'information. Nous nous concentrons sur des solutions simples :

— Une solution d'allocation de référence : Random. Dans cette solution l'allocation σ_R est calculée aléatoirement : pour l'application $\mathcal{A}_i$ nous choisissons uniformément n_i nœuds d'E/S parmi les N nœuds disponibles.
— Une solution *non-clairvoyante* Greedy-Non-Clairvoyant : cette solution n'utilise comme information pour chaque application que le nombre de nœuds d'E/S n_i. L'allocation σ_{NC} est calculée ainsi, à l'itération i :
    1. On trie les nœuds d'E/S en fonction du nombre d'applications qui les utilisent, c'est à dire pour le nœud j, la taille de l'ensemble $\{k | j \in \sigma_{NC}(k) \text{ et } k < i\}$.
    2. $\sigma_{NC}(i)$ reçoit les n_i nœuds les moins chargés (selon le tri précédent).
— Une solution clairvoyante Greedy-Clairvoyant : cette solution utilise pour chaque application le nombre de nœuds d'E/S n_i, ainsi que le ratio de temps d'E/S r_i. L'allocation

σ_C est calculée ainsi, à l'itération i :

1. On trie les nœuds d'E/S en fonction de leur charge moyenne, c'est à dire pour le nœud j, si les applications $i_1, i_2, \ldots, i_k$ sont alloués sur le nœud j, sa charge moyenne est définie par $r_{i_1} + \cdots + r_{i_k}$.
2. On alloue à l'application $\mathcal{A}_i$ les n_i nœuds les moins chargés (selon le tri précédent).

3. Environnement de test

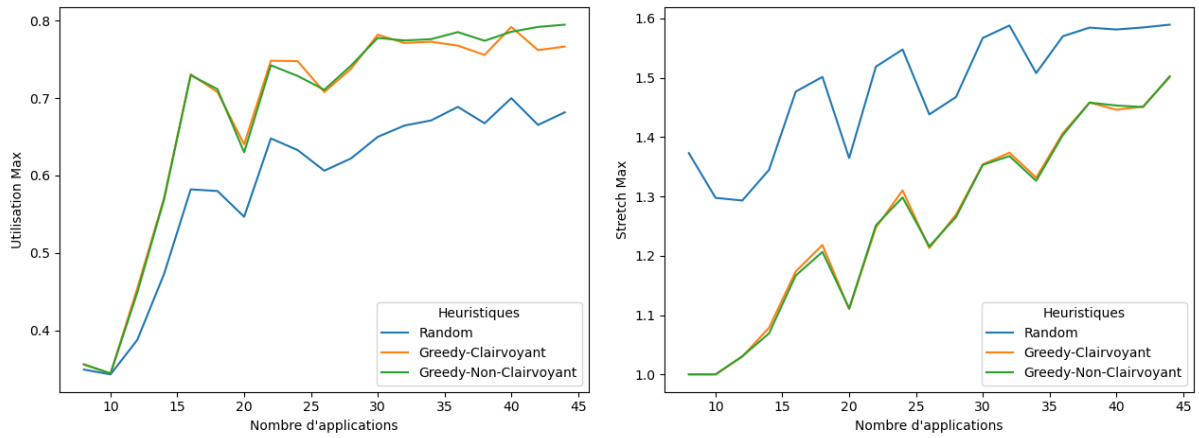
Pour notre première évaluation, nous avons développé un simulateur. Cela simplifie le processus et limite les ressources et temps nécessaires pour l'obtention des résultats. De plus, ce travail repose sur la possibilité de reconfigurer à souhait les accès aux différents nœuds E/S, ce qui n'est pas facile d'émuler en pratique. Une autre avantage d'un simulateur est qu'il permet de tester des scénarios à une échelle plus grande que les machines mises à notre disposition. Enfin, le simulateur permet d'isoler les comportements des applications qui nous intéresse sans être bruité par des interférences ou des spécificités de la machine. Notre simulateur est disponible sur `https://gitlab.inria.fr/hpc_io/ionode_simulator`.
Dans cette partie nous nous intéresserons dans un premier temps au mode de fonctionnement du simulateur (3.1). Puis nous parlerons de la génération d'applications et de la configuration utilisée dans nos expériences (3.2).

3.1. Simulateur de transfert E/S

Le simulateur est conçu dans le but qui nous intéresse pour cette recherche : savoir quelle pénalité subirait une application en cas de partage des ressources de la couche de transfert des E/S. Le simulateur reçoit un ensemble d'applications avec un système simplifié de temps sous forme d'itération. Chaque itération représente une unité de temps où l'application peut faire exclusivement des calcul/communications, des opérations E/S ou être en pause (« idle »). Chaque application aura besoin d'un nombre de nœuds E/S, déterminé lors d'une étape précédente.
Les applications sont distribuées aux différents nœuds E/S au début de la simulation. L'accès aux ressources E/S est exclusive : une seule application à la fois peut effectuer une opération E/S sur un nœud. Cette politique d'accès aux ressources a été montrée plus optimale que l'accès simultané [**?**] en terme de maximisation de la bande passante globale des E/S de la machine. Ainsi si plusieurs applications essayent d'accéder à la même ressource au même moment, une seule sera sélectionnée pour faire de l'E/S.
L'application qui attend depuis le plus longtemps sera prioritaire, sur le modèle Premier Arrivé, Premier Sorti (FIFO). Sinon le choix est aléatoire afin de ne pas introduire d'optimisation « biaisant » nos résultats.
Le simulateur implémente les comportements d'E/S bloquantes ou non-bloquantes, discutées dans la Section 2.1. Pourtant, dans ce papier, à cause de la limitation d'espace, nous ne discutons que les expériences avec la stratégie non-bloquante.

3.2. Configuration des expériences

Chaque application est générée sous forme d'un vecteur d'itérations de taille qui correspond à son temps d'exécution. Chaque itération est attribuée à calcul ou E/S de façon pseudo-aléatoire (en utilisant le générateur Mersenne Twister) en fonction de r_i, comme discuté dans la Section 2.1.
Les r_i des applications sont choisis de façon uniforme entre $0,05$ et 1, et les n_i entre 1 et 4. Les paramètres de chaque application sont générés indépendamment des autres. Nous simulons une machine équipée de 20 nœuds E/S.



(a) Utilisation des nœuds E/S

(b) Stretch

FIGURE 1 – Comparaison des algorithmes étudiés quand le nombre d'applications évolue. Chaque résultat est le maximum des résultats des 150 simulations. Les axes y sont différents.

L'occupation théorique de la machine occ sera

$$\text{occ} = \frac{\#\text{app} \times \bar{N} \times \bar{R}}{\#\text{ion}} \tag{1}$$

avec #app le nombre d'applications et #ion le nombre de nœuds d'E/S. Pour notre configuration, l'occupation de 100% serait aux alentours de 17 applications.
Dans nos expériences, toutes les applications ont la même durée de 100 itérations. Pour chaque configuration d'expérience, nous générons 150 charges de travail (ensembles d'applications) et testons toutes les algorithmes avec les mêmes charges de travail.

4. Résultats

Nos premiers résultats sont présentés sur la Figure 1. Nous pouvons constater que nos solutions Greedy-Clairvoyant et Greedy-Non-Clairvoyant sont plus efficaces que Random en utilisation des ressources d'E/S (Figure 1a) et en stretch (Figure 1b). Ce phénomène s'explique par une meilleure répartition de la charge sur les nœuds. La Figure 2 montre la disparité de la charge sur les nœuds quand Random est utilisé.
La courbe d'utilisation monte rapidement dans un premier temps jusqu'au point des 100% d'occupation de la machine (Équation 1), vers les 18 applications. Au-delà de ce point, les collisions deviennent plus probables et sont impossibles à éviter complètement.
Une chose intéressante à remarquer est que les résultats avec les algorithmes Greedy-Non-Clairvoyant et Greedy-Clairvoyant sont très similaires. Cela veut dire qu'ajouter des informations sur les applications (leur r_i) n'améliore pas les décisions prises. Par contre, nous voyons que l'utilisation d'une heuristique qui travaille pour équilibrer les charges des nœuds E/S, même si très simple, apporte déjà une augmentation significative des performances. L'application la plus impactée par le partage est 20% moins pénalisé avec les heuristiques optimisé par rapport à l'heuristique Random. Aussi, le temps perdu (*idle time*) est réduit de 15% par rapport au Random sur le même ensemble d'applications. Cela montre une optimisation des nœuds E/S de la machine. Ce résultat est encourageant pour la suite de notre recherche.
Afin d'améliorer la version globale du simulateur nous avons essayez de limiter l'effet domino

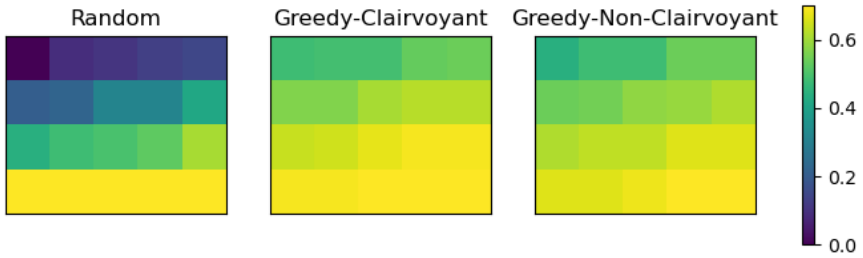


FIGURE 2 – Heatmap de l'utilisation des nœuds E/S avec 20 applications. Pour chaque algorithme, les 20 nœuds sont montrés. Pour cela, une des 150 expériences a été sélectionné aléatoirement.

entre les applications. En effet une application peut par effet de bord arrêter une application même si elle ne partage pas de nœuds d'E/S avec elle. L'idée est alors de former des sous groupes imperméables de nœuds E/S sur la machine. Chaque sous ensemble de nœud va posséder un sous ensemble d'applications à ordonnancer. Ainsi des frontières logicielles sont bâtis entre les différents groupes d'applications, ce qui devrait limiter les effets de bords.

5. Conclusion

Dans ce papier nous avons étudié le problème du placement des applications sur les nœuds d'E/S sur les super-calculateurs, en considérant le cas où le nombre optimal de nœuds est pré-déterminé pour chaque application. Plus spécifiquement, nous avons étudié comment ces applications pouvaient partager les nœuds d'E/S disponibles afin de minimiser les collisions d'E/S.
Pour cela nous avons proposé deux heuristiques, Greedy-Non-Clairvoyant et Greedy-Clairvoyant. Les résultats préliminaires montrent que ces deux heuristiques obtiennent de meilleurs résultats par rapport au placement Random. C'est selon nous une démonstration importante pour promouvoir une meilleure utilisation des centres de calculs.
La comparaison des algorithmes montre que celui utilisant plus d'information sur les applications (Greedy-Clairvoyant) n'apporte pas d'améliorations significatives des performances. Cela indique que même avec un minimum d'information, disponible au moment de l'exécution, est suffisant pour apporter de meilleurs résultats.
Beaucoup de nouvelles hypothèses sont soulevées par ce travail. Premièrement, une heuristique mettant en avant la séparation ou l'agrégation d'applications en fonction de leur nombre de nœuds d'E/S pourrait être intéressante puisque les applications ayant beaucoup de nœuds ont une plus grande chance d'entrer en collision et donc d'être pénalisé par le partage. Ensuite, l'expérimentation sur des applications fonctionnant avec des phases de calcul et d'E/S peut se révéler intéressante.

Bibliographie

1. Bez (J. L.), Miranda (A.), Nou (R.), Boito (F. Z.), Cortes (T.) et Navaux (P.). – Arbitration Policies for On-Demand User-Level I/O Forwarding on HPC Platforms. – In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 577–586, mai 2021. – ISSN : 1530-2075.
2. Carretero (J.), Jeannot (E.), Pallez (G.), Singh (D. E.) et Vidal (N.). – Mapping and scheduling HPC applications for optimizing I/O. – In *Proceedings of the 34th ACM International Conference on Supercomputing*, pp. 1–12, Barcelona Spain, juin 2020. ACM.
3. Dorier (M.), Ibrahim (S.), Antoniu (G.) et Ross (R.). – Omnisc'io : a grammar-based approach to spatial and temporal I/O patterns prediction. – In *SC'14 : Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 623–634. IEEE, 2014.
4. Gainaru (A.), Aupy (G.), Benoit (A.), Cappello (F.), Robert (Y.) et Snir (M.). – Scheduling the I/O of HPC applications under congestion. – In *2015 IEEE International Parallel and Distributed Processing Symposium*, pp. 1013–1022. IEEE, 2015.
5. Ji (X.), Yang (B.), Zhang (T.), Ma (X.), Zhu (X.), Wang (X.), El-Sayed (N.), Zhai (J.), Liu (W.) et Xue (W.). – Automatic, Application-Aware I/O Forwarding Resource Allocation. p. 16.
6. Ji (X.), Yang (B.), Zhang (T.), Ma (X.), Zhu (X.), Wang (X.), El-Sayed (N.), Zhai (J.), Liu (W.) et Xue (W.). – Automatic, Application-Aware I/O Forwarding Resource Allocation. – In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pp. 265–279, Boston, MA, 2019. USENIX Association.
7. Ohta (K.), Kimpe (D.), Cope (J.), Iskra (K.), Ross (R.) et Ishikawa (Y.). – Optimization techniques at the I/O forwarding layer. – In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, pp. 312–321, 2010.
8. Vishwan (V.), Hereld (M.), Iskra (K.), Kimpe (D.), Morozov (V.), Papka (M. E.), Ross (R.) et Yoshii (K.). – Accelerating I/O Forwarding in IBM Blue Gene/P Systems. – In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–10, New Orleans, LA, USA, novembre 2010. IEEE.
9. Xu (W.), Lu (Y.), Li (Q.), Zhou (E.), Song (Z.), Dong (Y.), Zhang (W.), Wei (D.), Zhang (X.), Chen (H.), Xing (J.) et Yuan (Y.). – Hybrid hierarchy storage system in MilkyWay-2 supercomputer. *Frontiers of Computer Science*, vol. 8, 2014, pp. 367–377.